Leszek Chmielewski
Pracownia Modelowania Systemów
Komputerowego Wspomagania
ORT IPPT PAN

PROPOZYCJA STANDARDU NAZW ZMIENNYCH W PROGRAMACH ANALIZY OBRAZU

1. Wstęp

Prace nad zagadnieniami analizy obrazu prowadzą do powstawania dużych programów, realizujących bardzo różne zadania - począwszy od analizy niskiego poziomu, gdzie wykonuje się przeważnie operacje arytmetyczne na tablicach, a skończywszy na analizie wysokiego poziomu, gdzie przeważają operacje logiczne na listach i grafach. Struktury danych dla takich programów są bardzo rozbudowane. Obmyślanie deklaracji takich stuktur stwarza potrzebę przyjęcia jakichś stałych zasad, które pomogłyby uniknąć zaplątania się w gąszczu typów, zmiennych i ich deskryptorów. (Mówiąc o zmiennych będziemy mieli na myśli również ich typy, a deskryptory będziemy nazywać po prostu nazwami.)

O ile nie ma wątpliwości co do tego, że postać struktur danych jest czymś ważnym i wymagającym przyjęcia odpowiednich ustaleń pomiędzy współautorami oprogramowania, o tyle sprawa umowy co do nazw zmiennych zwykle bywa pozostawiana na uboczu. Każdy programista dodając nową zmienną do zbioru zmiennych globalnych nadaje jej nazwę w sposób arbitralny.

Mając za sobą wiele godzin spędzonych na tak mało twórczej czynności, jak rozszyfrowywanie programów napisanych prez kogoś innego, których użycie, modyfikacja lub konserwacja przypadła mi

w udziale, postanowiłem poświęcić więcej uwagi prozaicznej, lecz nie pozbawionej znaczenia sprawie uporządkowania nazewnictwa w programach analizy obrazu budowanych przez członków i współpracowników Pracowni Modelowania Systemów Komputerowego Wspomagania ORT IPPT PAN. Proponuję mianowicie pewien standard, według którego byłyby tworzone przynajmniej nazwy globalnych zmiennych i typów w tych programach, czyli tych zmiennych, do których mają dostęp wszyscy autorzy kolejnych procedur.

Moralne poparcie w moim zamierzeniu czerpałem z bardzo cennej książki Dennie Van Tassela pt. "Praktyka Programowania" [6]. Zacytuję tu jedno zdanie z tej książki: "Używanie standardowych skrótów i standardowych nazw zmiennych jest szczególnie korzystne, gdy wielu programistów pracuje nad dużym programem." [6, rozdz. 1: Styl programowania. Standardowe skróty. Str. 23].

Proponowany standard został zastosowany w dużym i stale rozwijanym programie LOOK [2], rozpoznającym płaskie przedmioty, który jest budowany w PMSKW od 1987 roku.

2. Przesłanki

Nazwa zmiennej powinna mówić, czym jest ta zmienna. W dużym programie wystarczające określenia zmiennych i typów mogą składać się z wielu słów, wśród których można wyróżnić podstawowy rzeczownik, oraz rzeczowniki i przymiotniki pełniące rolę przydawek, np.: *plama obrazu, wskaźnik do pierwszego obiektu, plamka konturu obiektu, typ zbioru plam, górny wiersz obrazu* itp. Pisanie wszystkich słów w całości, nawet jeśli dopuszcza to kompilator używanego języka, prowadzi do zbyt długich, niezręcznych nazw.

Zachodzi więc potrzeba stosowania skrótów. Ponieważ wiele wyrazów powtarza się w nazwach bardzo często, w naturalny sposób nasuwa się pomysł przyjęcia dla nich ujednoliconych skrótów i budowania z nich nazw jako skrótowców - słów złożonych ze skrótów innych słów.

Koncepcję ustalenia standardowych skrótów często używanych wyrazów opisuje Dennie Van Tassel [6, str. 24]. Takie ustalenie jest potrzebne przed podjęciem wspólnych prac przez zespół programistów. Van Tassel podaje nawet (za pracą [4]) prostą metodę skracania wyrazów angielskich.

Metodę zastępowania nazw instrukcji systemowych mnemonicznymi symbolami, zawierającymi ustaloną liczbę liter, zastosowano w wielu systemach operacyjnych, jak np. George 3 [1] i RSX-11 [3]. Bez trudu rozpoznajemy też zawartość plików według trzyliterowych rozszerzeń ich nazw, np. w systemach MS-DOS [5] i RSX-11. Niektóre skróty są nawet powszechnie stosowane w całym środowisku programistów (OBJ, ASM, FOR itp.).

Wartość tych wypróbowanych systemów symboli polega głównie na tym, że zastosowane w nich skróty łatwo kojarzą się ze słowami, które oznaczają. Zwróćmy uwagę, że najlepiej przyjęły się symbole trzyliterowe, które są dość krótkie, a jeszcze pozwalają przybliżyć brzmienie całych wyrazów.

3. Wybór języka

Języki programowania opierają się na słownictwie angielskim. Brak polskich liter w zestawach dopuszczalnych znaków alfanumerycznych sprawia, że nie jesteśmy pewni, czy "czesc" znaczy "*część*" (part) czy "*cześć*" (hello), a "prog" - to skrót od "*program*" (program) czy "*próg*" (threshold).

Jeśli założymy, że programista zna język angielski, co zwykle jest prawdą, to spośród dwóch kawałków kodu, przytoczonych poniżej, może mu być trudniej przeczytać ten drugi, w którym słowo "*then*" [ðen] jest pomiędzy słowami "*wartosc*" a "*użyj*", co w tym połączeniu nie jest najłatwiejsze do wymówienia:

```
(1)
read(value) ;
if value > threshold
  then Use(value)
  else value := 0. ;
```

```
(2)
read(wartosc)
if wartosc > prog
  then Uzyj(wartosc)
  else wartosc := 0. ;
```

Nie można więc odmówić racji stwierdzeniu, że lepsze jest konsekwentne używanie języka angielskiego, niż mieszanie słów pochodzących z dwóch różnych języków. Języka polskiego można użyć jedynie w komentarzach.

Przedstawione stwierdzenia dotyczą również komputerów, mających w zestawie znaków ekranowych polskie litery (Mazovia, IPACO). Tych liter nie czytają jednak kompilatory.

4. Przykład

Weźmy przykładowy program w Pascalu.

Załóżmy, że w tablicy picture (lub pic), o indeksach - współrzędnych verse (lub vrs) i column (clm) mamy kolory, czyli jasności, kolejnych plamek (pixels - pix) obrazu. Na obrazie jest jeden obiekt o jednym konturze, zapisanym w wektorze set_of_pixels_in_contour (setpixctr), w ten sposób, że w każdym wyrazie tego wektora zapisano jedną plamkę konturu, tzn. jej wiersz i kolumnę. Szukamy plamki należącej do konturu, której kolor jest najjaśniejszy.

Poniżej zamieszczono dwie wersje zapisu programu. Nazwy w pierwszej wersji są tak dokładne, że nawet niepotrzebne są komentarze o nich. Jednak cały tekst programu jest wyjątkowo rozwlekły.

W drugiej wersji, ze skrótami, zapis stał się zwięzły, a nazwy nie straciły nic ze swej zawartości informacyjnej. Do zbudowania 14 nazw użyto 9 trzyliterowych symboli.

Przykładowe programy są ilustracją zasad tworzenia nazw za pomocą skrótów. Będą one przedstawione w następnym rozdziale.

```
program Example1 (input, output) ;

{stale potrzebne w deklaracjach zakresow indeksow}
const
  maximum_number_of_colours          =   15 ;
  maximum_number_of_verses           =  127 ;
  maximum_number_of_columns          =  255 ;
  maximum_number_of_pixels_in_contour = 1024 ;
```

```
type
  type_of_colour   = 0..maximum_number_of_colours ;
  type_of_index_of_pixel_in_contour
                   = 1..maximum_number_of_pixels_in_contour ;
  type_of_pixel    = record
                       verse  : 0..maximum_number_of_verses ;
                       column : 0..maximum_number_of_columns ;
                     end ;
  type_of_set_of_pixels_in_contour
                = array[1..maximum_number_of_pixels_in_contour]
                        of type_of_pixel ;
  type_of_picture = array[0..maximum_number_of_verses,
                          0..maximum_number_of_columns]
                          of type_of_colour ;

var
  picture : type_of_picture ;
  set_of_pixels_in_contour
          : type_of_set_of_pixels_in_contour ;
  index_of_pixel_in_contour,
  number_of_pixels_in_contour
          : type_of_index_of_pixel_in_contour ;
  pixel_of_maximum_colour   : type_of_pixel ;
  maximum_colour_in_contour : type_of_colour ;


procedure GetPicture(var picture : type_of_picture) ;

{Procedura pobierajaca obraz z kamery i umieszczajaca go w
 tablicy "picture" }

begin  {...}  end {of GetPicture} ;


procedure FindContour(picture : type_of_picture ;
                      var set_of_pixels_in_contour
                        : type_of_set_of_pixels_in_contour ;
                      var number_of_pixels_in_contour
                        : type_of_index_of_pixel_in_contour) ;

{Procedura znajdujaca kontur na obrazie, umieszczajaca go w
 tablicy "set_of_pixels_in_contour", liczaca plamki w kon-
 turze i umieszczajaca ich liczbe w zmiennej
 "number_of_pixels_in_contour" }

begin  {...}  end {of FindContour} ;


procedure OutputResults ;

{Procedura przekazujaca wyniki na zewnatrz}

begin  {...}  end {of OutputResults} ;
```

```
begin {of Example1}

  GetPicture(picture) ;
  FindContour(picture, set_of_pixels_in_contour,
              number_of_pixels_in_contour          ) ;

  {poszukiwanie plamki o najjasniejszym kolorze w konturze}

  maximum_colour_in_contour := 0 ;

  for index_of_pixel_in_contour
                    := 1 to number_of_pixels_in_contour do
    if (picture
      [set_of_pixels_in_contour[index_of_pixel_in_contour]
      .verse,
       set_of_pixels_in_contour[index_of_pixel_in_contour]
      .column]  > maximum_colour_in_contour)
    then

    begin

      pixel_of_maximum_colour.verse
        := set_of_pixels_in_contour[index_of_pixel_in_contour]
        .verse ;
      pixel_of_maximum_colour.column
        := set_of_pixels_in_contour[index_of_pixel_in_contour]
        .column ;
      maximum_colour_in_contour := picture
        [pixel_of_maximum_colour.verse,
         pixel_of_maximum_colour.column] ;

    end {of "if" clause and "for" loop} ;

  OutputResults ;

end {of Example1} .



program Example2 (input, output) ;

{skroty uzyte w tekscie programu:
 1. clm : column ;              6. mxn : maximum number ;
 2. col : colour ;              7. num : (actual) number ;
 3. ctr : contour ;             8. pix : pixel ;.
 4. ind : (current) index ;     9. vrs : verse ;
 5. max : maximum .                                    }
```

```
{stałe potrzebne w deklaracjach zakresów indeksów}
const
  mxncol     =   15 ;
  mxnvrs     =  127 ;
  mxnclm     =  255 ;
  mxnpixctr  = 1024 ;

type
  typcol        = 0..mxncol ;
  typindpixctr  = 1..mxnpixctr ;
  typpix        = record
                    vrs : 0..mxnvrs ;
                    clm : 0..mxnclm ;
                  end ;
  typsetpixctr  = array[1..mxnpixctr] of typpix ;
  typpic        = array[0..mxnvrs,0..mxnclm] of typcol ;

var
  pic        : typpic ;
  setpixctr  : typsetpixctr ;
  indpixctr,
  numpixctr  : typindpixctr ;
  pixmaxcol  : typpix ;
  maxcolctr  : typcol ;


procedure GetPicture(var pic : typpic) ;

{Procedura pobierająca obraz z kamery i umieszczająca go w
 tablicy "pic" }

begin  {...}  end {of GetPicture} ;


procedure FindContour(pic : typpic ;
                      var setpixctr : typsetpixctr ;
                      var numpixctr : typindpixctr   ) ;

{Procedura znajdująca kontur na obrazie, umieszczająca go w
 tablicy "setpixctr", licząca plamki w konturze i umieszcza-
 jąca ich liczbę w zmiennej "numpixctr" }

begin  {...}  end {of FindContour} ;


procedure OutputResults ;

{Procedura przekazująca wyniki na zewnątrz}

begin  {...}  end {of OutputResults} ;
```

```
begin {of Example2}

  GetPicture(pic) ;
  FindContour(pic, setpixctr, numpixctr) ;

  {poszukiwanie plamki o najjasniejszym kolorze w konturze}

  maxcolctr := 0 ;

  for indpixctr := 1 to numpixctr do
    if (pic[setpixctr[indpixctr].vrs,
            setpixctr[indpixctr].clm]  > maxcolctr)
    then

    begin

      pixmaxcol.vrs := setpixctr[indpixctr].vrs ;
      pixmaxcol.clm := setpixctr[indpixctr].clm ;
      maxcolctr     := pic[pixmaxcol.vrs,pixmaxcol.clm] ;

    end {of "if" clause and "for" loop} ;

  OutputResults ;

end {of Example2} .
```

5. Zasady

Obecnie zostaną omówione proponowane zasady tworzenia nazw zmiennych i ich typów w programach analizy obrazu, przez łączenie standardowych symboli - skrótów wyrazów opisujących te zmienne. Zastosowanie większości tych zasad ilustrują przykłady podane w poprzednim rozdziale.

1. Skróty są trzyliterowe.

Jest to kompromis pomiędzy długością a zrozumiałością skrótów.

2. Unika się łączenia więcej niż trzech skrótów w jednej nazwie.

3. W przypadku wątpliwości co do kolejności skrótów przyjmuje się taką kolejność, jakby pomiędzy słowami, które zastępują skróty, znajdowały się przyimki "of" lub "in".

W ten sposób najbardziej wyróżniające słowo nazwy zwykle znajduje się na początku, co umożliwia rozróżnienie nazw przez te kompilatory, które biorą pod uwagę tylko kilka początkowych liter.

W podanym przykładzie zastosowano kilka przedrostków : *typ*, *set*, *ind*, *num* itp. Kolejne zasady dotyczą przedrostków stosowanych do określania zakresów typów okrojonych, czyli używanych głównie w indeksach i wymiarach tablic.

4. Stałe określające zakresy typów okrojonych mają przedrostki *mnn* (minimum number) i *mxn* (maximum number).

Zamiast *mnn* najczęściej występuje liczba 0 lub 1.

5. Zmienne określające rzeczywiste rozmiary tablic mają przedrostek *num* ((actual) number).

W ten sposób indeks tablicy może przybierać wartości z zakresu <*mnn*,*num*> (najczęściej <1,*num*> lub <0,*num*>).

6. Bieżące indeksy mają przedrostek *ind* ((current) index).

7. Zmienne z przedrostkami *ind* i *num* są typu *typind*... .

Spójrzmy na drugą wersję programu w rozdziale 4. Tablica setpixctr ma rozmiar 1..mxnpixctr. Taki sam jest zakres zmiennych indpixctr i numpixctr, obydwóch tego samego typu typindpixctr. Pierwsza z nich to zmieniający się indeks aktualnie rozpatrywanej plamki konturu, a druga, to ustalona w procedurze FindContour, stała liczba plamek w konturze.

8. Jeśli w danej chwili rozpatrujemy podzakres całego zakresu zmiennej typu okrojonego, to jego ograniczniki mają przedrostki *min* i *max*.

Gdybyśmy rozpatrywali okno obrazu (*win* - window), to jego skrajne wiersze i kolumny nazwiemy *minrswin*, *maxrswin*, *mincimwin*, *maxcimwin*.

Na końcu artykułu zamieszczony jest słownik skrótów - symboli wyrazów użytych dotychczas w deklaracjach programu LOOK [2]. Umożliwia on zrozumienie zastosowania wszystkich globalnych i większości lokalnych zmiennych tego programu.

Plik DICT.TXT zawierający słownik jest napisany tak, aby sortowanie go standardową makroinstrukcją DOSa "sort" zachowywało alfabetyczną kolejność skrótów i położenie komentarzy.

Słownik DICT.TXT wraz z opisanymi tutaj zasadami stanowi propozycję standardowego sposobu tworzenia nazw zmiennych w programach analizy obrazu, pisanych przez członków i współpracowników Pracowni Modelowania Systemów Komputerowego Wspomagania ORT IPPT.

Przewiduje się, że użytkownicy będą rozwijać standard, dodając do słownika nowe skróty wyrazów, lecz raczej bez wprowadzania istotnych zmian do zasad, które zostały przedstawione powyżej.

6. Przewidywanie sprzeciwów

Sprzeciwy zawsze występują wtedy, gdy wykonawcy nie decydują sami o wszystkim, co dotyczy ich pracy. Programiści są często ludźmi o silnie rozwiniętym poczuciu niezależności; jednak pisanie dużych programów wymaga współpracy. W zstępującym procesie programowania strukturalnego dużo decyzji podejmuje się przed rozdzieleniem pracy kooperantom. Jeśli struktury zmiennych globalnych trzeba ustalić na początku, to lepiej to zrobić w sposób systematyczny, wprowadzając tym samym pewien standard.

Standaryzacja sposobu nazywania zmiennych służy ułatwieniu zapisywania programów, jak również czytania ich przez ludzi, czyli wzajemnego komunikowania się programistów. Rozwiązując sprawę formy komunikatu, pozwala poświęcić całą uwagę jego treści. Sprawa ma się podobnie, jak z umową o graficznym układzie tekstu programu. Stosowanie wcięć i odstępów oraz wstawianie komentarzy pomaga w poprawnym formułowaniu programów i pozwala łatwiej je potem zrozumieć. Przeciwko przyjęciu takiego jednolitego stylu pisania wysuwano zarzut, że może on krępować swobodę myślenia. Zarzut ten okazał się jednak nieuzasadniony i styl powszechnie się przyjął.

Dlatego sądzę, że jedynym właściwym argumentem przeciwko zaakceptowaniu tego standardu, o którym była tutaj mowa, mogłaby być tylko propozycja wprowadzenia innego, lepszego standardu.

7. Słownik: plik DICT.TXT

```
000
aa0aaaaaaaaaaaa Slowniczek skrotow mnemonicznych aaaaaaaaaaaaa
aa1 sluzacych do tworzenia nazw zmiennych globalnych i ich
aa2 typow w programach analizy obrazu. Wszystkie skroty sa
aa3 trzyliterowe i pochodza od slow angielskich.
aa4  Jezeli kilka skrotow w jednej nazwie oznacza rzeczow-
aa5 niki i powstaje watpliwosc co do ich kolejnosci w naz-
aa6 wie, to nalezy je tak ustawiac, jakby pomiedzy slowami
aa7 byl przyimek 'of' lub 'in', np.  typpic =  'type' 'of'
aa8 'picture', czyli typ obiektu 'obraz'.
aa9aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
abs : absolute              : absolutny       -> rel
ang : angle                 : kat
are : area                  : pole
arr : array                 : tablica         -> set,vec,rec
aux : auxiliary             : pomocniczy      -> tmp,buf
bcg : background            : tlo
blb : blob                  : plama
buf : buffer                : bufor           -> aux,tmp
cen : centre, centroid      : srodek, centroid
chd : child                 : dziecko         -> par,nbr,pre,nex
cla : class                 : klasa
clm : column                : kolumna         -> vrs
cnd : candidate             : kandydat
col : colour                : kolor
crd : coordinate            : wspolrzedna
ctr : contour               : kontur
```

dfl	default	domyslny	
dst	distance	odleglosc	
dyn	dynamic	dynamiczny	
fea	feature	cecha	
fil	file	plik	
fir	first	pierwszy	-> las,pre,nex
fra	frame	ramka	
ful	full	pelny, calkowity	
hgt	height	wysokosc	-> wdt
hol	hole	dziura	
hor	horizontal	poziomy	-> ver
ind	current index	biezacy indeks	-> mnn,mxn,num
inp	input	wejscie	-> out
inr	inner	wewnetrzny	-> otr
las	last	ostatni	-> fir,pre,nex
len	length	dlugosc	
lev	level	poziom	
lft	left	lewy	-> rht,upr,lwr
lin	line	linia	
lns	lens	obiektyw	
lst	list	lista	-> set,arr,vec
lwr	lower	dolny, nizszy	-> upr,lft,rht
mag	magnification	powiekszenie	
mai	main	glowny	
max	maximum	maksimum	-> min
min	minimum	minimum	-> max
mnn	minimum number	najmniejsza liczba	-> mxn,num,ind
msg	message	komunikat	
mxn	maximum number	najwieksza liczba	-> mnn,num,ind
nam	name	nazwa	
nbr	neighbour	sasiad	-> par,chd
nex	next	nastepny	-> pre,fir,las
num	actual number	rzeczywista liczba	-> ind,mxn,mnn
obj	object	obiekt	
otr	outer	zewnetrzny	-> inr
out	output	wyjscie	-> inp
par	parent	rodzic	-> chd,nbr
pat	pattern	wzorzec	-> rpp
per	perimeter	obwod	
pic	picture	obraz	-> scr,win
pix	pixel	plamka (element obrazu)	
pnt	point	punkt	-> sec
pre	previous	poprzedni	-> nex,fir,las
prt	prototype	prototyp (klasy)	
ptr	pointer	wskaznik	
rcg	recognition	rozpoznawanie	
rec	record	rekord	-> arr,vec,set
rel	relative	wzgledny	-> abs
rht	right	prawy	-> lft,upr,lwr
run	run	ciag (w kodowaniu)	
scr	screen	ekran	-> win,pic
sec	section point	punkt przeciecia	-> pnt
set	set	zbior	-> arr,vec,rec
sta	start	start, poczatek	

```
str : string          : lancuch
sum : sum             : suma
tmp : temporary       : tymczasowy      -> aux,buf
typ : type            : typ
upr : upper           : wyzszy, gorny   -> lwr,lft,rht
vec : vector          : wektor          -> set,arr,rec
ver : vertical        : pionowy         -> hor
vie : view            : widok
vrs : verse           : wiersz          -> clm
wdt : width           : szerokosc       -> hgt
win : window          : okno            -> scr,pic
zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz
```

LITERATURA

1. S. Borak, J. Klaczak, S. Korczak, Z. Płoski, *System operacyjny George 3*, WNT, Warszawa 1981.
2. L. Chmielewski, W. Kosiński, *Oprogramowanie do rozpoznawania płaskich, nie stykających się obiektów*, Sprawozadanie naukowe, PMSKW, ORT, IPPT PAN, 1987.
3. J. Deminet, *System operacyjny RSX-11*, WNT, Warszawa 1987.
4. Michael Jackson, Datamation 1967.
5. *MS-DOS Operating System, User's Guide*, MicroSoft Corp., 1981.
6. D. Van Tassel, *Praktyka programowania*, WNT, Warszawa 1978. (tłum. z ang.: *Program Style, Design, Efficiency, Debugging and Testing*, Prentice-Hall, Englewood Cliffs, 1978.)